Augmenting Recurrent Neural Networks Resilience by Dropout

Davide Bacciu, *Member, IEEE,* and Francesco Crecchi

***Abstract*—The paper discusses the simple idea that dropout regularization can be used to efficiently induce resiliency to missing inputs at prediction time in a generic neural network. We show how the approach can be effective on tasks where imputation strategies often fail, namely involving recurrent neural networks and scenarios where whole sequences of input observations are missing. The experimental analysis provides an assessment of the accuracy-resiliency tradeoff in multiple recurrent models, including reservoir computing methods, and comprising real-world ambient intelligence and biomedical time series.**

***Index Terms*—dropout, recurrent neural networks, deep learning, missing inputs, dependable machine learning**

I. Introduction

Decades of research on Artificial Neural Networks (ANNs) have promulgated the idea that ANNs are intrinsically sensitive to missing/incomplete inputs at prediction time [1], [2], [3], [4]. Studies on dependable ANNs [5], [6] show that a neural network cannot be considered intrinsically fault tolerant and it is not possible to induce complete error masking after a fault occurred, even in presence of learning. Specific methodologies and neural architecture have thus been proposed to enforce fault tolerance, but mostly restricted to failures in hidden neurons [7].

This paper tackles the rather basic research question of how we can make neural networks robust, "by design", to incomplete inputs at prediction time. The large majority of the works in literature addresses missing inputs solely with respect to information that is missing at training time. In this context, dealing with missing information amounts to finding the best strategies to impute values of certain features which are missing in a subset of the training samples [8]. In this brief paper, we assume to be able to train a neural network using complete data while part of the inputs may become unavailable during the deployment phase. This is a problem seldom addressed in literature and, yet, strongly impacting the dependability of a deployed neural model, especially when the data sources feeding the model are not reliable such as in mobile computing and Internet-of-Things (IoT) applications, where information sources can disappear due to communication or power issues.

The most popular approach for dealing with missing inputs at test time is based on completing the data using (more or less) sophisticated imputation methods ranging from statistical methods (e.g., mean imputation, regression imputation, multiple imputation and hot/cold deck imputation) to machine learning based approaches (e.g., k-nearest neighbor imputation, neural-based imputation, etc.). In [9], for instance, a spatial-temporal replacement scheme is proposed for a fuzzy Adaptive Resonance Theory (ART) neural network used for anomaly detection over a Wireless Sensor Network (WSN). Here missing input information related to a WSN node is imputed based on a majority voting between the readings of the devices that are spatially closer to the failing one. Similarly, [10] uses a k-Nearest Neighbor (kNN) to replace missing sensor values with spatially and temporally correlated sensor values. This approach has a major drawback in the fact that it assumes that sensor devices are of homogenous type and that it is possible to find a spatially/temporally related sensor, of the same type as the failing one, to replace its measurements. An alternative approach to the problem is based on fitting a probability distribution of the missing features given the observable inputs, e.g. by using a Parzen window estimator [11], and to use it to sample replacement measurements for the missing information [12], [13]. In [14], it is proposed a discriminative solution that uses a RNN to model static (i.e. non sequential) problems while recurrent layers are used to estimate the values of the missing features.

A rather different approach to the problem is that of designing a model which is intrinsically able to deal with missing inputs, which is typically tackled by resorting to ensemble architectures. In this approach, several models are fit at training time, each on a subset of the original inputs, so that during prediction with missing features, only those models which did not use those particular features for training are queried. The set of results is then summarized to produce a single output (e.g., using a majority voting scheme [15]). In [16], a set of MLPs is created and each MLP classifies based on each possible unique combination of complete features, in order to cover the complete range of attributes with missing values. The major drawback of this method is that it requires a huge number of neurons when multiple combinations of incomplete attributes are presented. To reduce the computational cost, [17] proposed to train an arbitrary number of classifiers, each fit on a random subset of the features, without guarantees of coverage for all possible combinations of missing features. The main drawback of ensemble methods is certainly their computational cost, since they need training, querying and aggregating results for a number of sub-models which is exponential in the number of independent input sources, hence making their real-world use impractical for non-trivial scenarios.

The approach put forward in this brief paper originates from a well-known interpretation of the Dropout regularization technique [18] as an ensembling method. We build on the fact that training a parent network with Dropout restricted to input units is equivalent to training a pseudo-ensemble [19] of child subnetworks, each missing variable combinations of the input features. By this means, it ensues a trained network that is inherently more robust to missing inputs, without incurring in significant computational overheads at training and at prediction time. The idea of Dropout acting as a regularizer for input noise (such as missing features) at prediction time is not novel. However, this is the first work discussing the systematic use of Dropout as an effective and efficient means to produce neural networks resilient to missing inputs. We provide an orderly experimental assessment of the effect of this technique on the resiliency of different neural models as well as of the trade-off between resiliency and accuracy (in absence of faults). Further, we show that the effect of input Dropout is not equivalent to that of data augmentation targeted at enforcing robustness to missing inputs.

In our analysis, we focus on recurrent neural architectures, motivated by two observations. First, Recurrent Neural Networks (RNNs) are specifically designed to handle sequential data, for which is difficult to define adequate imputation strategies as this would require the capability of generating a coherent sequence of related observations (rather than a single independent sample as in the static vectorial case). The second aspect is related with a popular usage scenario for RNNs, comprising the realization of classification and regression tasks on multivariate time series collected by networks of pervasively distributed environmental and personal sensor devices, such as in ambient intelligence and IoT. Such deployments involve information generated by networks of loosely connected devices, which are often battery-operated and communicating through wireless channels with little quality of service guarantees. In such scenarios, it is not unlikely to have to deal with missing information due to the brittle information sources involved, which can become unavailable due to communication problems, power failure and hardware problems.

We show that the proposed approach in sound and general enough to be applied to various RNN models, ranging from standard RNNs to gated recurrent networks, such as the Long Short Term Memory (LSTM) [20]. Further, we discuss how the model can be applied even

to randomized neural networks from the Reservoir Computing (RC) paradigm [21], where the recurrent layer is a randomly initialized and non-adaptive dynamic memory of the input. We show how the effect of input Dropout propagates through such non-adaptive layer inducing changes in the output weights (which is the only adaptive part of the network), which again make the model more robust to missing inputs.

A preliminary version of this work has appeared as a conference paper in [22], where the term DropIn is used to denote the application of Dropout to the input units for enforcing missing input robustness only on a RC network. This brief paper extends the work in [22] including more learning models (vanilla RNN and gated RNN), further and more challenging benchmarks (including a safety critical application on intensive care unit data) as well as more systematic characterization of the approach with respect to pseudo-ensemble and data augmentation methods. To this end, we also compare the performance of the DropIn scheme with respect to an equivalent data augmentation scheme, following the ideas in [23].

The remainder of the paper is structured as follows: Section II describes the DropIn approach and its pseudo-ensamble and data augmentation interpretation. Section III provides an empirical assessment of the proposed technique on three real-world datasets and different RNN models. Section IV concludes the paper.

II. Missing Inputs Resiliency and Dropout

A. DropIn - Enforcing Robustness to Missing Inputs by Design

We discuss how unit-wise Dropout [18] applied to the input neurons of the network can be used to efficiently simulate the presence of missing input patterns during training. We put forward the idea that this technique, which we refer to as DropIn, can be a principled approach to make neural networks robust to missing inputs at prediction time. Briefly, Dropout is a regularization technique for neural network layers based on the initial intuition that unit co-adaptation can be prevented by by randomly disconnecting a subset of them from the network during training. Dropout acts also as an ensembling technique, by efficiently combining an exponential number of neural network architectures (with respect to the number of neurons) corresponding to the thinned networks obtained by dropping the random neuron subsets.

Leveraging on this pseudo-ensemble [19] properties of Dropout, DropIn provides the benefits of a committee machine similar to [17] without the need to define an ad-hoc committee architecture nor inducing any increase in the parametrization and complexity of the original model. DropIn works by straightforward application of Dropout restricted to the input units. In summary, for each training sample the algorithm randomly determines which input features are kept. Each input unit is retained with a fixed probability p, independently from other neurons, or is dropped with probability $1-p$. A dropped out input neuron is temporarily removed from the network, along with all its outgoing connections: in practice, this amounts to zeroing the corresponding unit output. At test time, the network thinning process is approximated as in standard Dropout [18] by using the original un-thinned network with weights scaled by the retention probability p.

Intuitively, the retention probability p is associated with the proportion of missing inputs we would like the network to be prepared to handle. In practice, the value of p might need to be chosen in a problem dependent way, according to the amount of redundancy in the input information or following the accuracy constraints characterizing the task. In general, one can expect that lower values of p will make the network more prone to accommodate higher proportions of missing inputs; however, this typically comes at the cost of an overall reduced performance when all inputs are present (see the results of the experimental analysis in Section III). A fairly straightforward and practical way to handle this aspect would be that of maintaining a set of models trained with different retention probabilities to handle missing inputs of different intensities without depleting the overall performance. Nonetheless, the experimental assessment in Section III shows that a reasonable choice of p (i.e. considering to lose no more than 20% of the input units) provides excellent tradeoffs between robustness and accuracy in absence of missing inputs and across different tasks.

The DropIn approach is general, as it can be applied to any neural network model for which Dropout applies. Application to RNN and sequential data requires a single caution, that is that of masking the randomly chosen input units for the whole duration of the current sample-sequence. In fact, changing the input drop mask during the processing of a sequence might prevent the model from being able to capture relevant causal dependencies between its composing elements. To provide an indication of the generality of the DropIn approach, in the remainder of this paper, we apply it to RNN models with different complexities and capabilities. In particular, we consider two fully adaptive recurrent models, that are Elman's *Simple Recurrent Networks* (SRNs) [24] and the *Long Short Term Memory* (LSTM) gated networks [20]. Both models can be trained by Stochastic Gradient Descent (SGD) via *Backpropagation Through Time* (BPTT) [25]. The stochastic nature of SGD fits well the masking scheme required by DropIn, where unit masks are sampled once for each sequence on which training is performed. A third neural architecture considered in this work is the Echo State Networks (ESNs) [21]: this model is based on consistently different assumptions with respect to both SRNs and LSTMs, which require some careful considerations when applying the DropIn technique. First, the ESN is based on a randomly initialized and non-adaptive recurrent layer (with recurrent weights $\mathbf{W}_h$), directly attached to the input layer through connections with non-adaptive weights $\mathbf{W}_{in}$. The only adaptive elements of the network are the output connections $\mathbf{W}_{out}$. This is a challenging condition for testing the effect of the DropIn technique, as no direct regularization effect can be observed on the connections subject to dropout (which are themselves fixed and non-adaptive). Further, in order to be incorporated in the network, the effect of DropIn must propagate throughout the reservoir all the way to the output units, which is the only trained part of the network and hence the only part where the effect of DropIn can be recorded. Note that, apart from the preliminary version of this paper [22], this is the first work in which Dropout techniques are being used in RC.

The application of DropIn to ESN requires a modification of its standard training scheme, as this is based on the batch minimization of a linear least square problem, which is not compatible with the iterative scheme required by DropIn. It is straightforward to note that, in order to make the network robust to missing inputs, we need the ESN to process the single training sequences multiple times, each time with a fixed probability $1-p$ of independently missing the single inputs. To achieve this, it is sufficient to abandon the batch least square minimization algorithm in favour of the Recurrent Least Squares (RLS) for ESN training described in [21]. The modified version of this algorithm with the introduction of DropIn is described by the pseudo-code in Algorithm 1 (an interested reader can find further details in [22]).

B. Dropout, Ensambling and Data Augmentation

Despite its apparent simplicity and heuristic nature, Dropout has been shown to root its effectiveness on a robust interpretation as pseudo-ensembling and data augmentation technique. Here we briefly

Algorithm 1 DropIn Training of an ESN by RLS

Require: A dataset of L pairs of input-output sequences $(\mathbf{u}_1, \mathbf{y}_1^*), \ldots, (\mathbf{u}_L, \mathbf{y}_L^*)$; a properly initialized ESN model; a forgetting factor λ; a regularization term δ and an input unit retention probability p.
Init $\mathbf{S}^{-1}(0) = \delta^{-1} \cdot \mathbf{I}$
$n = 0$
while learning not converged **do**
  Shuffle sequence order
  for $i = 1$ to L **do**
    Compute masked inputs vector $\mathbf{M}$ using p
    for $t = 1$ **to** T_j **do**
      $n = n + 1$
      Mask current inputs $\mathbf{u}_i(t)$ using $\mathbf{M}$ obtaining $\tilde{\mathbf{u}}_i(t)$
      Compute reservoir activation

$$\mathbf{x}_i(t) = tanh(\mathbf{W}_{in}\tilde{\mathbf{u}}_i(t) + \mathbf{W}_h\mathbf{x}_i(t-1))$$

      Compute readout activation

$$\mathbf{y}_i(t) = \mathbf{W}_{out}\mathbf{x}_i(t)$$

      Compute error $\mathbf{e}(n) = (\mathbf{y}_i^*(t) - \mathbf{y}_i(t))$
      $\mathbf{\Phi}(n) = S^{-1}(n-1) \cdot \mathbf{x}_i(t)$
      $\mathbf{K}(n) = \mathbf{\Phi}(n) \cdot (\lambda + \mathbf{\Phi}(n) \cdot \mathbf{x}_i(t))^{-1}$
      $\mathbf{S}^{-1}(n) = \lambda^{-1}(\mathbf{S}^{-1}(n-1) - \mathbf{K}(n) \cdot \mathbf{\Phi}(n))$
      $\mathbf{W}_{out}(n) = \mathbf{W}_{out}(n-1) + (\mathbf{K}(n) \cdot e(n))^T$
    end for
  end for
end while
return $\mathbf{W}_{out}$

review this aspects from the perspective of using Dropout to enforce missing input resilience.

A pseudo-ensemble method [19] is a (possibly infinite) collection of child models $f_\Theta(x;\xi)$ spawned from a parent model f_Θ by perturbing it according to some noise-process $\xi \sim p_\xi$. Differently from classical ensemble methods, pseudo-ensemble involve noise in the model space rather than in the input space thus obtaining the benefits of ensemble methods without paying their computational cost since all sub-models share their parameters. The goal of learning with pseudo-ensembles is to produce models robust to perturbation, which can be generally stated as

$$\min_{\Theta} \mathop{\mathbb{E}}_{(x,y)\sim p_{x,y}} \mathop{\mathbb{E}}_{\xi\sim p_\xi} \mathcal{L}(f_\Theta(x;\xi), y) \tag{1}$$

where $(x, y) \sim p_{x,y}$ is an observation-label pair drawn from the data distribution and $\mathcal{L}(\hat{y}, y)$ is the loss associated to prediction $\hat{y}$ and ground-truth y. The pseudo-ensemble approach allows much freedom in the definition of the noise process p_ξ and the mechanisms by which ξ perturbs the parent model f_Θ. In these terms, the missing data problem fits the pseudo-ensemble framework considering an unknown noise process which has introduced *missingness* in data. DropIn trains the neural model as a pseudo-ensemble of subnetworks subject to the missing-input noise model. The parameters shared by the sub-networks, through their common source network, are learned to minimize the expected loss of the individual subnetworks subject to such noise. Hence, it trivially follows that the resulting pseudo-ensemble network will have a performance coherent with such expectation when subject to a similar noise process at test time (i.e. when some inputs are missing).

Recent work [23] characterizes Dropout from a different perspective, as a domain-oblivious data augmentation technique in input space. They project the Dropout noise (also of internal units) back into the input space, thereby generating augmented version of the training data, and showing that training a deterministic network on the augmented samples yields to comparable results. Moreover, the analysis suggests that Dropout induces a very limited set of transformations in the input space and that richer sets of transformations can be desirable. Following this idea by [23], in the experimental analysis, we introduce a data augmentation method to emulate the DropIn effects but in the input space. This is intended to study if the DropIn effect can be trivially reconstructed by an appropriate data augumentation policy. Having chosen the target input retention probability p, the original dataset is augmented by masking training data through the corresponding DropIn mask. This enriches the original data with missing input features, so that during training the model can learn to cope with partial information. It is worth to notice that accounting for all possible missing features pattern would let the problem to be intractable due to exponential complexity of the augmentation procedure. Considering only masking using p reduces the data augmentation complexity to linear in the number of missing features and it is coherent with the assumptions of the DropIn noise.

III. Experimental Results

We provide an experimental assessment of the effect of DropIn in three real-world datasets from Ambient Assisted Living (AAL) and biomedical applications. These two applications scenarios have been chosen for the presence of heterogenous sensor sources, different levels of data redundancy and scale and complexity of the learning task. For the sake of this work, we consider missing inputs at test time while we assume that a sufficient amount of complete training data is available. All datasets are public and available for download.

A. Data and Experimental Setup

The objective of the experimental assessment is twofold. On the one hand, we confront the resilience of different RNN models to missing test inputs with respect to a DropIn equivalent. On the other hand, we compare the performance of DropIn against the data augmentation technique described in Sect. II-B. In both settings, we test performance for a varying input-retention probability $p \in \{0.8, 0.5, 0.3\}$.

Three RNN types are considered in this experimental analysis: SRN, LSTM and ESN, which have all been assessed both with and without DropIn training. All models have been regularized using alternative schemes to Dropout (e.g. L2 regularization) to avoid interfering with the assessment of the DropIn alone. A model selection scheme has been set-up as follows: first, for each dataset, a hold-out of 20% of the total sequences has been retained to create an external test set. A k-fold cross validation, with $k = 3$, has then been applied to the remaining data for parameter search. Parameter grids can be found in Table I. Once best parameters have been chosen, for each model, a baseline performance on a test set without missing inputs is provided. Then we show the effect of an increasing number of missing features, by performing test predictions with input features removed. For instance, the performance with a single missing input is obtained by removing a single input feature for each test sequence at a time, then the network prediction is computed and the process is iterated for all inputs and an average prediction performance is computed. Similarly, when testing multiple missing inputs, we perform predictions using all possible combinations of missing features and we then provide performance statistics averaged on all combinations.

The first dataset[1] concerns indoor human movement prediction [26] from homogenous input signals consisting in the Received Signal

[1]Available here: https://archive.ics.uci.edu/ml/datasets/Indoor+User+Movement\\+Prediction+from+RSS+data

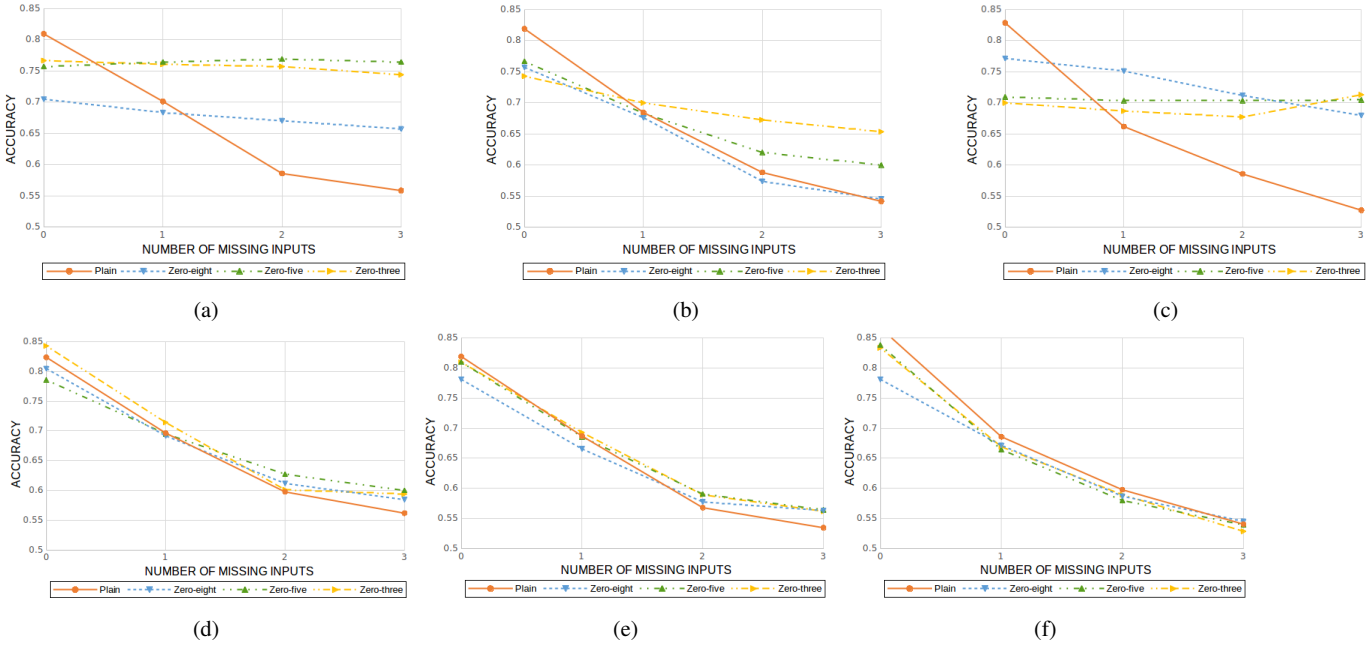

Fig. 1: Test set performance for the Movement Prediction Task. Plots show recurrent models (SRN, LSTM and LI-ESN, respectively) performance in case of DropIn application (a, b, c) and data augmentation technique (d, e ,f), varying the number of missing inputs.

RNN Type	Parameters
SRN, LSTM	$N \in [50, 100, 250]$ $\lambda \in [0.00001, 0.001, 0.1, 0]$
LI-ESN	$N_R \in [50, 100, 250, 500]$ $\alpha \in [0.1, 0.2, 0.3, 0.5, 1]$

TABLE I: RNN model parameters considered for model selection: N is recurrent layer size in dense networks, λ is the L2 regularization parameter, N_R is the reservoir size and α is the leaking-rate for ESN.

Strenght (RSS) of packets exchanged between a WSN mote worn by the user and four anchor devices deployed at fixed positions in the corners of rooms. We consider the *homogeneous* setup in [26], comprising 210 RSS sequences concerning trajectories of different users moving between two rooms couples separated by a hallway. The task requires to predict whether the user is going to change room or to stay in the current one and it is modelled as a binary classification problem and it is thus assessed by classification accuracy.

The second dataset, referred to as Kitchen Cleaning [27], [28], concerns an AAL scenario where a cleaning robot operates in a home environment. The dataset comprises information from 5 wireless presence sensors and concerning the location of the robot in the environment, as measured by its range-finder localization system (see [28] for details on the dataset). The task concerns the prediction of a success measure for the cleaning task, associated to the presence of the user in the kitchen while the robot its attempting to clean. Specifically, the target value will tend to 1 (success) when the user is outside of the kitchen, while it will approach 0 (likely to fail) when the user enters the kitchen. The performance here is assessed using *Mean Absolute Error* (MAE) as targets and outputs are computed for each element of the sequences.

The third dataset concerns prediction of mortality rates in *Intensive Care Units* (ICU) populations. The data set came from the *PhysioNet/CinC Challenge 2012* [29], held in 2012, to develop methods for *patient-specific* prediction of in-hospital mortality. We refer to the version of the dataset used by [30] in a study presenting an extension of a GRU network that can handle missing inputs at training time. For the sake of this work, we have extracted a subsample of 2799 subjects out of the original 4000 as these are characterized by the absence of missing measurements in 27 fields of the patient records (remember that in this work we do not consider missing inputs at training time). The features considered for our work are: Age, BUN, Creatinine, DiasABP, GCS, Gender, Glucose, HCO3, HCT, HR, Height, ICUType, K, Mg, NIDiasABP, NIMAP, NISysABP, Na, PaCO2, PaO2, Platelets, SysABP, Temp, Urine, WBC, Weight, pH. Note how this is a widely heterogenous dataset, comprising both timeseries information as well as static data. One-hot encoding has been applied to categorical features, i.e. Gender, ICUType, leading to a total of 31 input features. The extracted dataset has been preprocessed for input feature normalization and measurements have been hourly resampled. Following [30], performances for this task are measured by the *Area Under the (ROC) Curve* score (AUC).

B. Results and Analysis

Figure 1 depicts the test-set performance on the first task for the three recurrent models as the number of missing inputs increases. In this figure and in the following ones, the first row shows DropIn models' results while the second row shows the performance of the models trained with data augmentation. As a reference, in both rows, we always report the results of the plain model, i.e. one not using neither DropIn nor data augmentation.

The baseline results (i.e. first point of the solid line), which corresponds to testing on a complete input, show that models trained without DropIn achieve higher test accuracies with respect to DropIn models. This is not surprising as, for instance, a DropIn model with retention probability $p = 0.3$ on this task means, essentially, that it has been trained using only information from a single input randomly selected each time from the available four. However, the performance reduction of DropIn models is limited when all inputs are available, while DropIn shows clear benefits when dealing with missing inputs. In fact, the performance of plain models drops considerably (to about 70%) already when a single input is missing. On the contrary, DropIn models shows a smoother performance drop, remaining well above

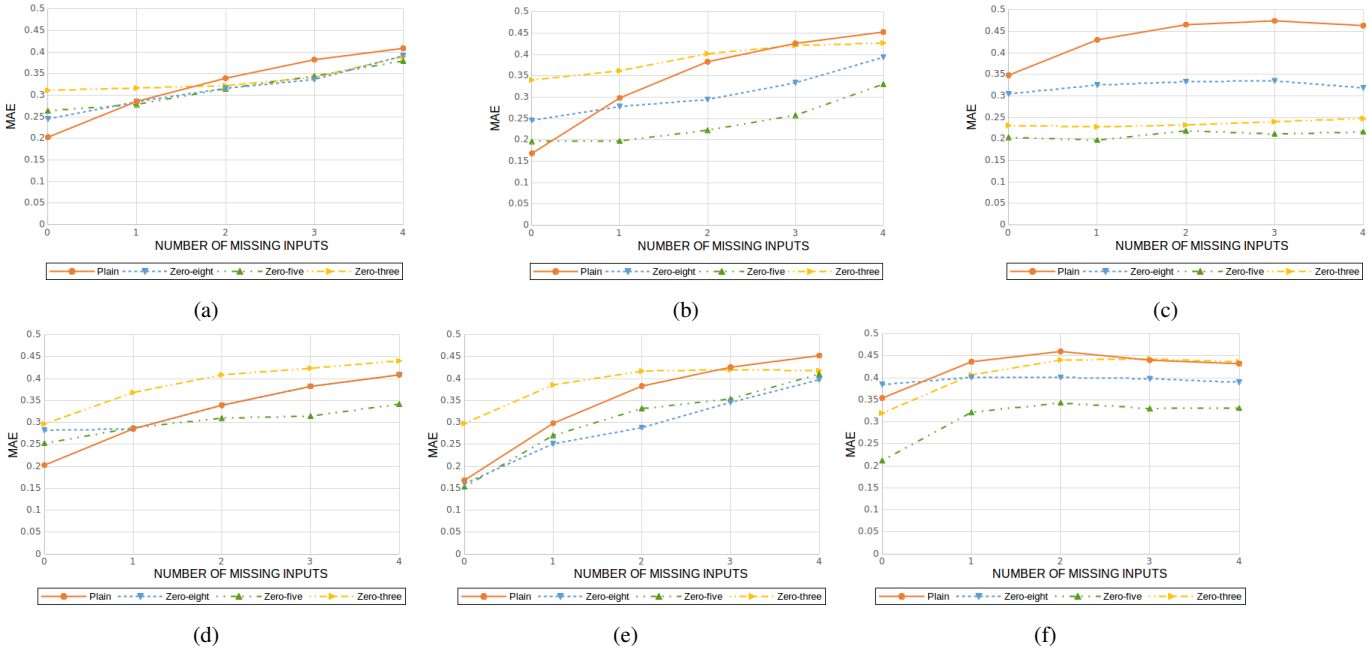

Fig. 2: Test set performance for the Kitchen Cleaning Task. Plots show recurrent models (SRN, LSTM and LI-ESN, respectively) performance in case of DropIn application (a, b, c) and data augmentation technique (d, e ,f), varying the number of missing inputs.

the standard model even when more inputs are lost. This is motivated by the fact that testing a DropIn model with $p = 0.8$ for predictions with one missing input, allows the network to work under the pseudo-ensemble training conditions, i.e. with a noise process masking on average one input each time. Interestingly enough the same advantage cannot be achieved by applying the data augmentation technique. The performance drop, in this case, seems equivalent to that of a plain model (no DropIn and no data-augmentation), having a steep decay even when just one input is missing. For this task thus, DropIn seems to behave significantly different from data augmentation, introducing a beneficial effect in terms of missing input resiliency for all the RNN types under consideration.

Figure 2 shows the results for the Kitchen Cleaning task: the beneficial effect of DropIn training is quite evident also in this task. Interestingly, also data augmentation has a positive effect on this task, leading to a smoother performance degradation than plain models. By confronting DropIn and data augmentation one can note different behaviours depending on the target learning model. In simpler networks (i.e. SRN and ESN) DropIn seems to yield to a better behaviour than data augmentation, with less marked performance variations depending on the amount of input retention probability applied at training time. On LSTM, on the other hand, the data augmentation techniques seems to be able to yield to more accurate models. This can be explained by the fact that explicit data augmentation might help increasing generalization of an highly parameterized model, such as LSTM, on this small sample size benchmark.

On the last benchmark, whose results are in Fig. 3, one can note a different behaviour. Only the DropIn technique seems to have a beneficial effect with respect to a plain model and only for the ESN case. This is the best scoring model on the task, despite having a non-adaptive recurrent part. The effect of DropIn in this case is particularly interesting as the input perturbation introduced by DropIn training somehow manages to transfer to the adaptive read-out units producing a beneficial effect already with 6 missing inputs out of 31. DropIn seems to have a slightly beneficial effect also on SRN and LSTM, but only in the extreme case where almost all of the inputs are missing. Data augmentation does not seem to provide any beneficial effect on the task, possibly due to the inputs being redundant and given the complexity of the predictive task.

IV. Conclusion

We have shown how a widely known regularization technique, Dropout, can be effectively used to train neural networks that are resilient to missing inputs at testing time. The proposed approach is simple and computationally efficient. It does not use any external or companion learning model to provide advanced input imputation functionalities. Rather it works on the model itself, making it robust by design with only a minimal change to the training process. The approach is also general and applicable to any neural model, even those for which we typically do not use Dropout regularization, such as reservoir computing models. We have also discussed the relationship between DropIn and data augmentation techniques, providing empirical evidence that DropIn is not equivalent to perform data augmentation in input space, at least from the point of view of the performance of the trained model. Further, one can expect that exhaustive data augmentation can quickly become computationally infeasible for non trivial input space sizes.

Acknowledgment

The work is partially funded by Italian Ministry of Education, University, and Research (MIUR) under project SIR 2014 LIST-IT (grant n. RBSI14STDE)

References

[1] A. Morris, L. Josifovski, H. Bourlard, M. Cooke, and P. Green, "A neural network for classification with incomplete data: application to robust asr," *Proc. ICSLP*, 2000.

[2] M. K. Markey and A. Patel, "Impact of missing data in training artificial neural networks for computer-aided diagnosis," in *Proc. of Int. Conf. on Machine Learning and Applications*, Dec 2004, pp. 351–354.

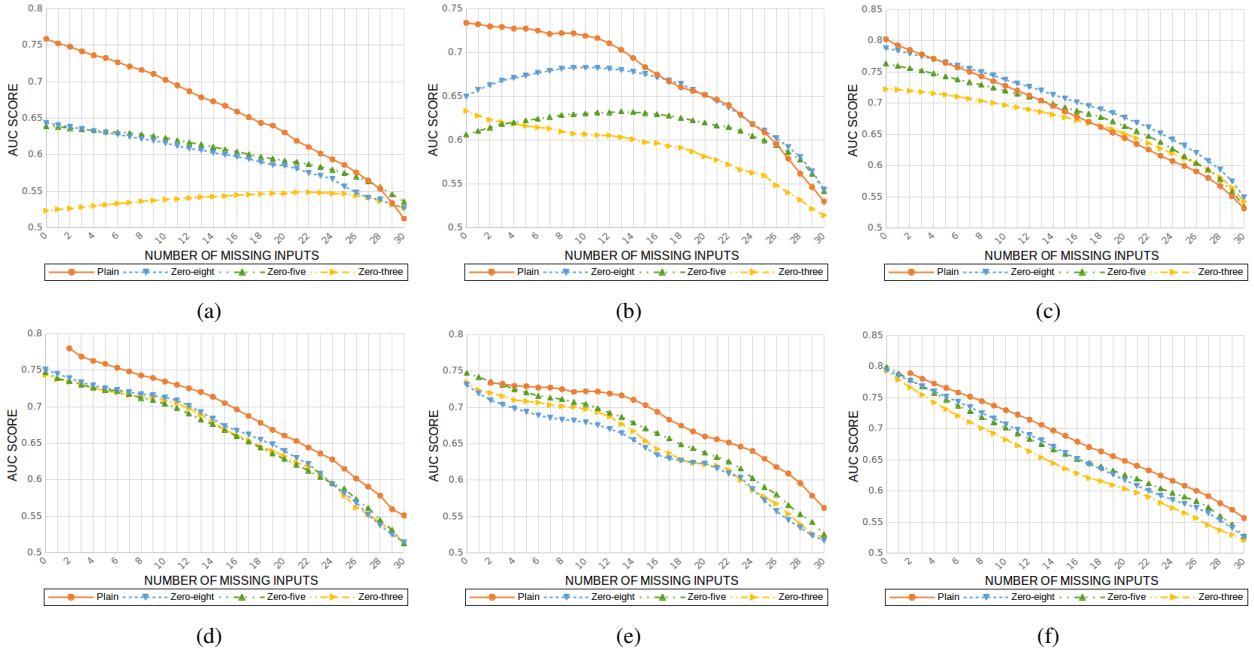

Fig. 3: Test set performance for the Physionet Mortality Prediction Task. Plots show recurrent models (SRN, LSTM and LI-ESN, respectively) performance in case of DropIn application (a, b, c) and data augmentation technique (d, e ,f), varying the number of missing inputs.

[3] M. L. Brown and J. F. Kros, "Data mining and the impact of missing data," *Industrial Management & Data Systems*, vol. 103, no. 8, pp. 611–621, 2003.

[4] G. M. Kashif, A. Tirusew, K. Yasir, and M. Mac, "Effect of missing data on performance of learning algorithms for hydrologic predictions: Implications to an imputation technique," *Water Resources Research*, vol. 43, no. 7, 2007.

[5] C. Alippi, V. Piuri, and M. Sami, "Sensitivity to errors in artificial neural networks: a behavioral approach," *IEEE Trans. on Circuits and Systems*, vol. 42, no. 6, pp. 358–361, Jun 1995.

[6] V. Piuri, "Analysis of fault tolerance in artificial neural networks," *Journal of Parallel and Distributed Computing*, vol. 61, no. 1, pp. 18 – 48, 2001.

[7] E. M. E. Mhamdi and R. Guerraoui, "When neurons fail," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2017, pp. 1028–1037.

[8] P. J. García-Laencina, J.-L. Sancho-Gómez, and A. R. Figueiras-Vidal, "Pattern classification with missing data: A review," *Neural Comput. Appl.*, vol. 19, no. 2, pp. 263–282.

[9] Y. Li and L. E. Parker, "Classification with missing data in a wireless sensor network," in *IEEE SoutheastCon 2008*, April 2008, pp. 533–538.

[10] ——, "Nearest neighbor imputation using spatialtemporal correlations in wireless sensor networks," *Information Fusion*, vol. 15, pp. 64 – 79, 2014.

[11] V. Tresp, R. Neuneier, and S. Ahmad, "Efficient methods for dealing with missing data in supervised learning," in *Advances in neural information processing systems*. Morgan Kaufmann, 1995, pp. 689–696.

[12] V. Tresp, S. Ahmad, and R. Neuneier, "Training neural networks with deficient data," in *Advances in neural information processing systems*. Morgan Kaufmann, 1994, pp. 128–128.

[13] V. Tresp and R. Hofmann, "Missing and noisy data in nonlinear time-series prediction," in *Proc. of IEEE Workshop on Neur. Netw. for Signal Proc*. IEEE, 1995, pp. 1–10.

[14] Y. Bengio and F. Gingras, "Recurrent neural networks for missing or asynchronous data," *Advances in neural information processing systems*, pp. 395–401, 1996.

[15] K. Jiang, H. Chen, and S. Yuan, "Classification for incomplete data using classifier ensembles," in *2005 International Conference on Neural Networks and Brain*, vol. 1, pp. 559–563.

[16] P. K. Sharpe and R. J. Solly, "Dealing with missing values in neural network-based diagnostic systems," *Neural Computing and Applications*, vol. 3, no. 2.

[17] S. Krause and R. Polikar, "An ensemble of classifiers approach for the missing feature problem," in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, vol. 1, pp. 553–558 vol.1.

[18] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958.

[19] P. Bachman, O. Alsharif, and D. Precup, "Learning with pseudo-ensembles," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., pp. 3365–3373.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[21] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, no. 5667, pp. 78–80.

[22] D. Bacciu, F. Crecchi, and D. Morelli, "DropIn: Making reservoir computing neural networks robust to missing inputs by dropout," in *Proc. of the 2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2080–2087.

[23] X. Bouthillier, K. Konda, P. Vincent, and R. Memisevic, "Dropout as data augmentation," *arXiv:1506.08700*, 2015.

[24] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211.

[25] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, oct 1990.

[26] D. Bacciu, P. Barsocchi, S. Chessa, C. Gallicchio, and A. Micheli, "An experimental characterization of reservoir computing in ambient assisted living applications," *Neural Computing and Applications*, vol. 24, no. 6, pp. 1451–1464.

[27] D. Bacciu, C. Gallicchio, A. Micheli, M. D. Rocco, and A. Saffiotti, "Learning context-aware mobile robot navigation in home environments," in *IISA 2014, The 5th International Conference on Information, Intelligence, Systems and Applications*, jul 2014, pp. 57–62.

[28] D. Bacciu, "Unsupervised feature selection for sensor time-series in pervasive computing applications," *Neural Computing and Applications*, vol. 27, no. 5, pp. 1077–1091, jul 2016.

[29] I. Silva, G. Moody, D. J. Scott, L. A. Celi, and R. G. Mark, "Predicting in-hospital mortality of ICU patients: The PhysioNet/computing in cardiology challenge 2012," *Computing in Cardiology*, vol. 39, pp. 245–248.

[30] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent neural networks for multivariate time series with missing values," *Nature Scientific Reports*, vol. 8.